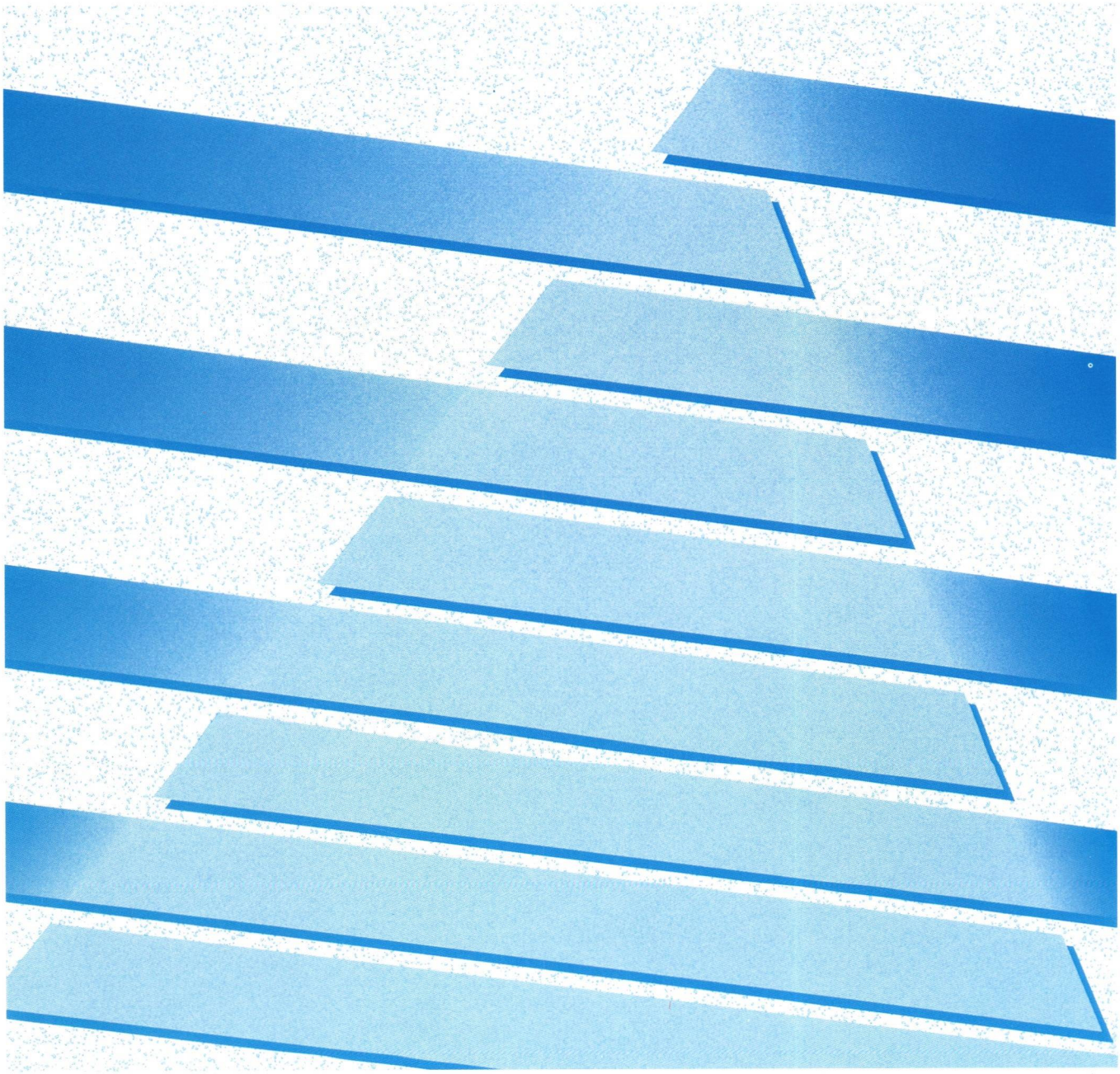
ALLEN-BRADLEY

ControlView™ Derived Tags

(Cat. No. 6190-DTM)

User Manual



Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, the Allen-Bradley Company, Inc. does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley Publication SGI-1.1, "Safety Guidelines for the Application, Installation and Maintenance of Solid State Control" (available from your local Allen-Bradley office) describes some important differences between solid-state equipment and electromechanical devices which should be taken into consideration when applying products such as those described in this publication.

Throughout this manual we use notes to make you aware of safety considerations:

ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss.

Attentions help you:

- identify a hazard
- avoid the hazard
- recognize the consequences

Important: Identifies information that is especially important for successful application and understanding of the product.

Summary of Changes

Changes from Release 2.0 to 3.0

The following changes have been made to the Derived Tags option and the Derived Tags User Manual since release 2.0:

For information on this new feature:	Refer to:	The feature appeared in:
Installation instructions for the Derived Tags option have been moved to the *ControlView Installation Manual.*	ControlView Installation Manual	software release 3.0

Preface

How To Use This Manual

This manual describes the features and capabilities of the Derived Tags option of the ControlView™ System.

The Derived Tags processor is used to continuously calculate complex equations, called expressions, and to store the results in the ControlView Current Value Database. These expressions can be composed of mathematical operations, tag values from the Current Value Database, IF-THEN-ELSE logic, and other special functions.

Results of the expressions are stored as local tags in the Current Value Database (CVD) and can be used by other applications such as Trending, Alarming, the Event Detector, and C-Toolkit programs.

For more information on local tags, refer to the *ControlView Core User Manual*, Chapter 3, *The Setup Menu.*

Conventions Used in This Manual

This manual follows the print conventions outlined in the *ControlView Core User Manual.*

Audience

Since the Derived Tags module is a part of ControlView, you should be familiar with ControlView and have the *ControlView Core User Manual* available for reference. A complete list of related publications is contained in that manual.

Defining Derived Tags

Chapter 1

Defining an Expression

Chapter 2

Derived Tags Commands

Appendix A

Index

Chapter 1

Defining Derived Tags

The Derived Tags processor continuously calculates expressions, and stores the results in the Current Value Database. These expressions can be composed of mathematical operations, tag values from the Current Value Database, IF-THEN-ELSE logic, and other special functions. Chapter 2, *Defining an Expression*, describes how to create expressions.

Local tags are used to store the result of the expression. This chapter explains how to define a local tag, using the Derived Tag Editor. Local tags are never updated with values from the programmable controller.

The Database Editor

For complete instructions on defining tags, see the *ControlView Core User Manual,* Chapter 2, *The Setup Menu.* To create a local tag:

1. Load the database editor and select a database.
2. Add a new analog or digital tag.
3. Specify the *Address Type* as "None".
4. Leave the *Address*, *Scan Class* and *Node* fields blank.
5. Enter an *Initial Value*. For analog tags, keep this between the Minimum and Maximum values. For digital tags, use the text entered as the *On* or *Off* label for the initial value.

Remember that for each derived tag you want to define, there must be a local tag in the database. Once the local tags have been created, go on to define the derived tags.

At the time of this release, string tags are not supported by this option.

The Derived Tag Editor

To create or edit a file of derived tags, choose *Edit Derived Tags* under Configure in the Setup Menu.

You are prompted to name a derived tag file. You may type in:

- the name of an existing file, to modify it

  If you have forgotten the name of a derived tag file, go to the Actions Menu and select *Start Derived Tags* under Start/Stop to display a popup list of available tag files.

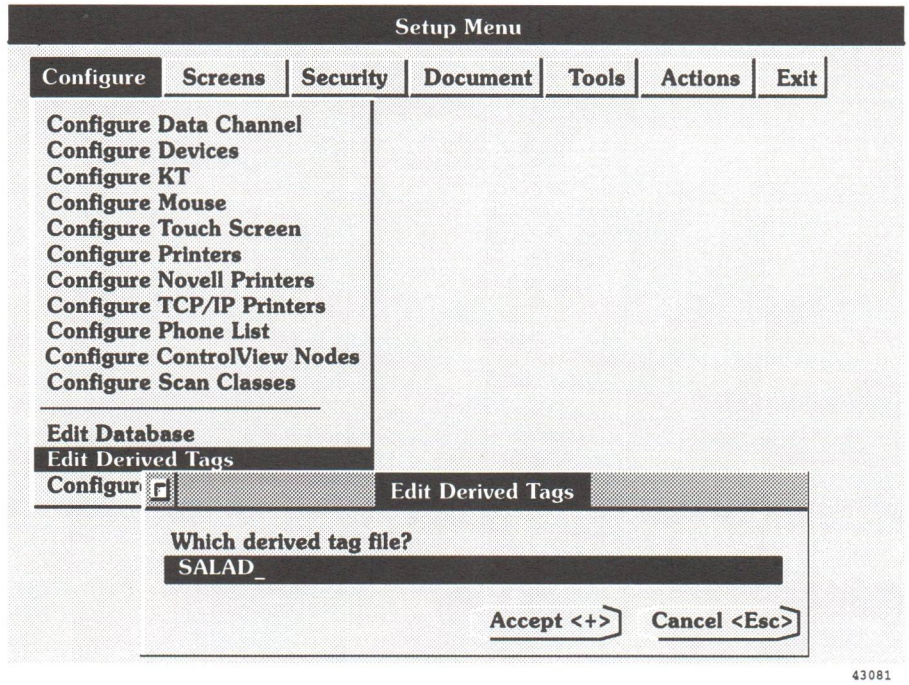
- a new name, to create a file

- leave the field blank, to use the default Derived Tag file called DTAGS

One of the features of the Derived Tag Editor is autoverification. When turned on, it checks to ensure that every derived tag specified actually exists as a local tag in your database. It also checks that every tag used in an expression also exists in the database. If you intend to add autoverification to a derived tag, load the database before you open the Derived Tag Editor. (Autoverification is set with *Setup* in the Derived Tags Editor.)

Important: Autoverification does *not* affect system performance. (However, turning off autoverification enables you to develop a derived tag list for a database that does not exist locally.)

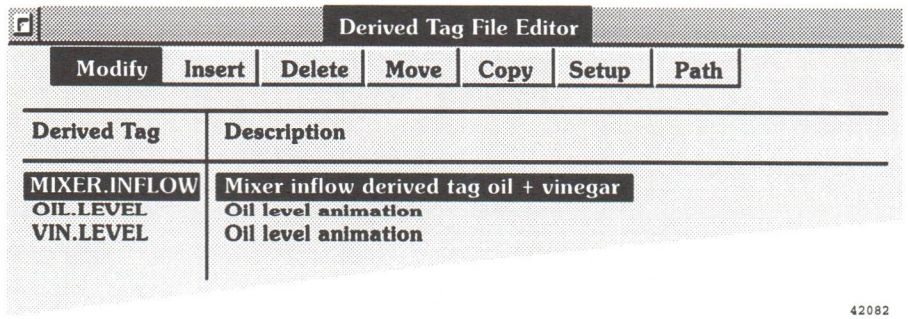
Figure 1.1
Starting the Derived Tag Editor



In the following sections, the menu items are described in left-to-right order.

If the file is empty, the first thing to do is choose *Insert* and define a derived tag.

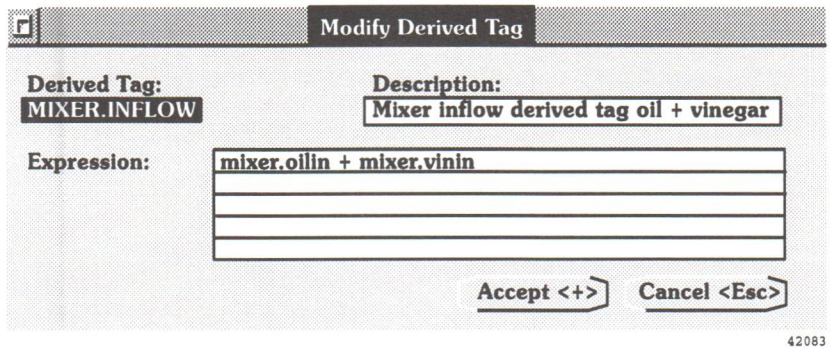
Figure 1.2
The Derived Tag File Editor



Modify

To edit a derived tag, highlight the tag name and choose *Modify*, or double-click on the derived tag name.

Figure 1.3
Modify Derived Tag Window



You can edit any of the three fields. For details on the *Derived Tag*, *Description* and *Expression* fields, see the description of Insert.

Save the changes with the *Accept* button or the + key.

Insert

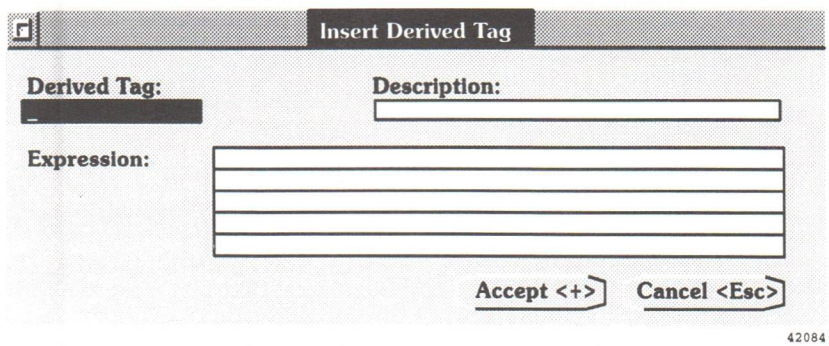
To create a new derived tag:

1. Determine where the new tag will go by highlighting a tag in the Derived Tag Editor. The new tag will appear below the selected one.

2. Choose *Insert* to add a new tag. The Insert Derived Tag window opens.

Important: Derived tags are processed in the order they are listed. So, if the tag you are about to create depends on the value of a tag in the list, then it should be placed after that tag.

Figure 1.4
Insert Derived Tag Window



- Derived Tag

  Type a tagname for a local tag which will hold the derived tag value.

- Description

  Enter a brief description to identify the derived tag. This description appears in the Derived Tag Editor and is displayed only for your information.

- Expression

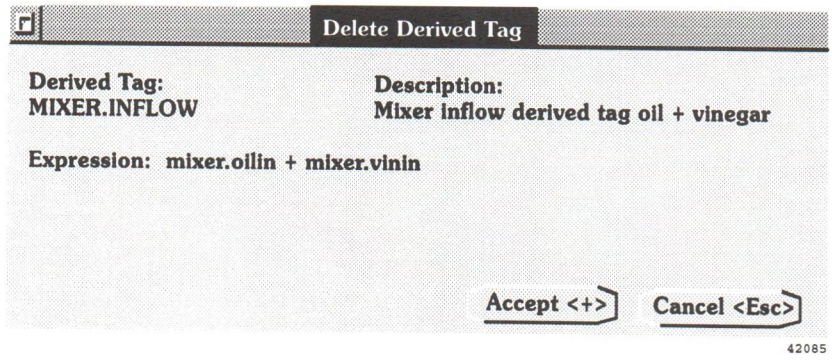
  Enter the expression which will create the derived tag value in this field. For details on how to use the Expression field, see Chapter 2, *Defining an Expression.*

Save the derived tag with the *Accept* button or the + key. The new definition will appear in the Derived Tag Editor.

Delete

To erase a derived tag definition, highlight the tag and choose *Delete*. The tag's definition appears in a window as illustrated in Figure 1.5. Confirm the deletion with the *Accept* button or the + key.

Figure 1.5
Delete Derived Tag Window



Move

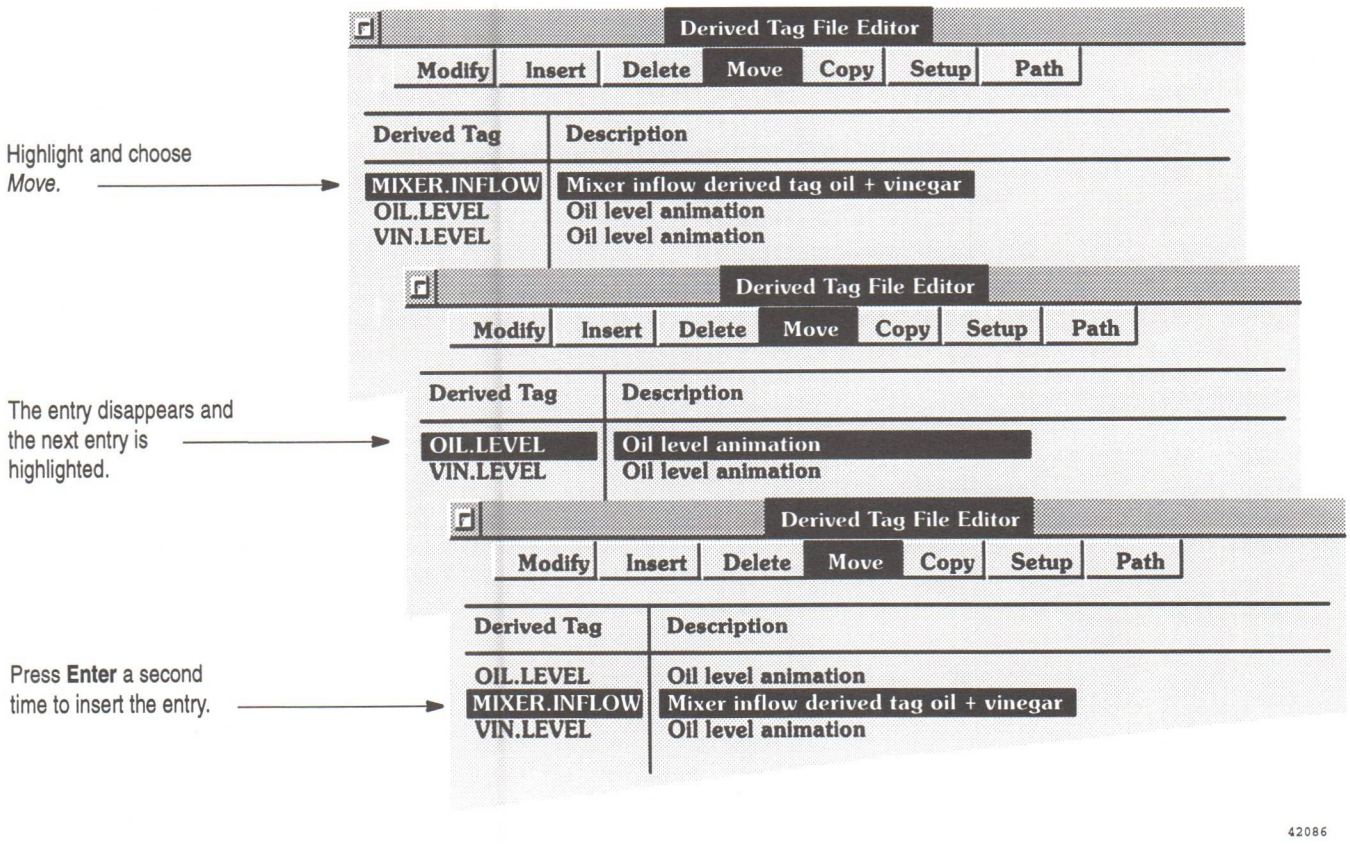
Derived tags are processed in the order they are listed. Therefore, if derived tag2 depends on the value of derived tag1, ensure tag1 is listed earlier, so that tag2 uses the most up-to-date value.

To change the order of the derived tag list, highlight a tag name and choose *Move*. The highlighted tag disappears from the list and a flashing blue bar appears over the next tag in the Derived Tag Editor.

To reposition the tag, highlight the new position and press **Enter**, or double-click on the desired destination. The tag will be placed below the highlighted entry.

Important: To move a tag to the top of the list, move it below the top entry, then move the top entry down one position.

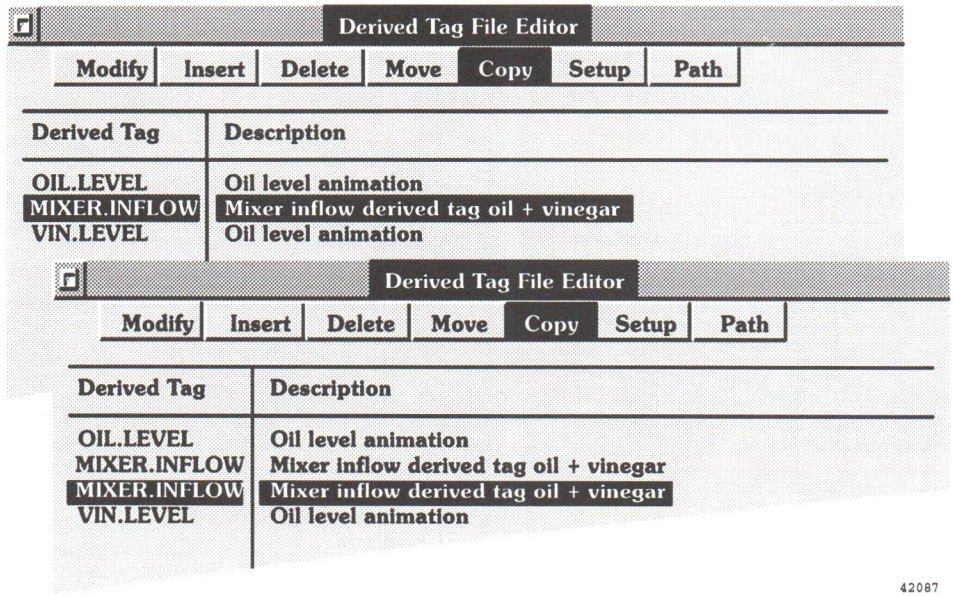
Figure 1.6
Moving a Derived Tag



Copy

To duplicate a derived tag definition, highlight the tag and choose *Copy*. An identical entry appears below the original.

Figure 1.7
Copying a Derived Tag

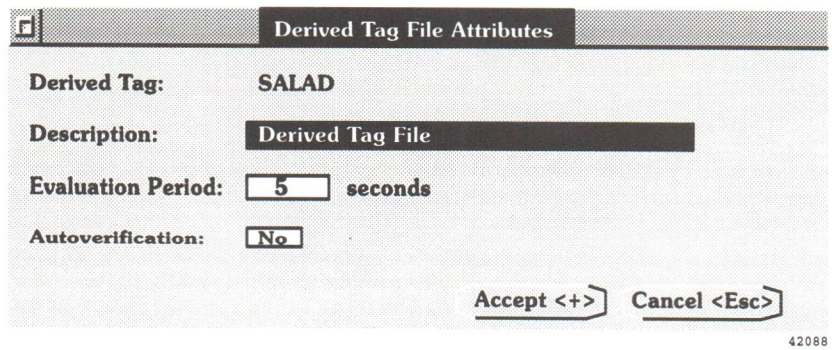


You can then move or modify the copy.

Setup

Choose *Setup* to define the characteristics of the derived tag file; the following window appears.

Figure 1.8
Derived Tag File Attributes Window



- Derived Tag File

  This field contains the name of the file that was originally loaded.

Important: Depending on the size of the expressions, each file can contain as many as 300 derived tags, accessing a maximum of 600 tags.

- Description

  Enter a brief description of the file to identify it. This explanation is used only for documentation purposes.

- Evaluation Period

  The derived tag processor continually cycles through the list of derived tags, evaluating each expression and updating the local tags. The frequency of this cycle, or Evaluation Period, ranges from 0 to 99,999,999 seconds.

  To complete this field, specify, in seconds, how often the processor should cycle through the list of tags.

  An Evaluation Period of 0 means the processor will cycle through the list as fast as possible without pausing. However, such speed can tie up the processor and impair system performance.

  If your expressions rely on tag values, remember that tags are assigned to scan classes. The scan class determines how often the tag is updated with new PLC® values. Keep the different scan classes in mind as you determine the derived tag's evaluation period.

  Set the evaluation period for the derived tags to a rate equal to or slower than the scan rate. If you set the rate faster, the expressions will be performing calculations using PLC values that haven't changed since the last evaluation, and the system will be burdened with needless processing. For more information on scan classes, see *ControlView Core User Manual*, Chapter 2, *The Setup Menu.*

- Autoverification

  When autoverification is on, the editor will make sure that all tags used in the expression exist in the database, and that the tag named as the derived tag is a local one. For autoverification to work, you must load the database before you call up the editor.

  Choose *YES* for autoverification, *NO* to turn it off.

Important: Immediately after autoverification is applied, the system slows down while the existing database is checked. After this initial database check, autoverification does *not* affect system performance.

Important: When you start the Derived Tag Editor from the menu, (instead of the command line) autoverification will always be turned off, regardless of the setting of this field.

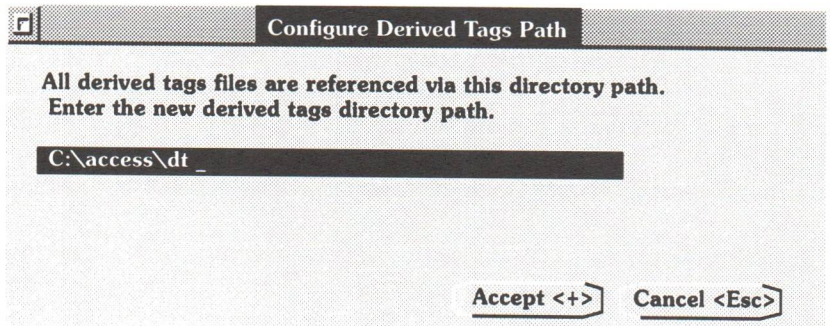
Path

Specify the DOS pathname where the derived tags file is located. The default directory is C:\ACCESS\DT where C is the drive where ControlView is installed.

Important: Whenever you configure a pathname, be sure to start with the drive letter. This is absolutely essential when running ControlView in a multi-drive environment.

Important: This path must be used by all derived tag files and must not be altered for individual files. The derived tag processor will only work from the specified path. Changing the path of an existing derived tag file, moves the file to the new location, deleting it from the old location.

Figure 1.9
Configure Derived Tags Path Window



Start the Derived Tag Processor

Before starting the Derived Tag Processor, load the database containing all the local tags and the tags used in the Derived Tag Expressions. To load the processor, choose *Start Derived Tags* under Start/Stop in the Actions menu. A list of the derived tags files will appear. Choose the file you wish to run.

Important: Only one file can be run at a time.

Stop the Derived Tag Processor

To stop the Derived Tag Processor, choose *Stop Derived Tags* under Start/Stop in the Actions menu.

Chapter 2

Defining an Expression

Defining an Expression

Sometimes the data gathered from remote drivers is only meaningful when you:

- combine it with other values
- compare it to other values
- create a cause - effect relationship with other values

Expressions allow you to create mathematical or logical combinations of data that return more meaningful values. The term refers to an entire equation whereas a segment of an expression is called a *statement.*

Expressions can be built from:

- tag values
- constants
- mathematical, relational, logical, and bitwise operators
- built-in functions
- IF-THEN-ELSE program logic

Important: All expressions return floating point values.

Example: Expressions vs Statements

```
(tag1 * tag2) AND (tag3 / 2)
```

is an expression

```
(tag3 / 2)
```

is a statement

Tag Values

A tag can be included as part of an expression, or can stand alone as the entire expression.

Constants

Using the number 123.45 as an example, a constant could have any of the following formats:

- integer (123)
- floating point value (123.45)
- scientific notation (1.2345E2)

Arithmetic Operators

The arithmetic operators are as follows:

Table 2.A
The Arithmetic Operators

Symbol	Operator	Example
+	addition	tag1 + tag2
–	subtraction	tag1 – tag2
*	multiplication	tag1 * tag2
/	division	tag1 / tag2
MOD, %	modulus (remainder)	tag1 % tag2
**	exponent	tag1 ** tag2

Important: The modulus operator is the remainder of one number divided by another.
For example, the remainder of 13 divided by 5 is 3; so 13 % 5 = 3.

Important: Expressions which attempt to divide a number by zero are ignored. No error message is given. Be sure that any tag value which is used as a divisor cannot at some point have a value of zero.

Example: Arithmetic Operators

For these examples, tag1 = 5 and tag2 = 7.

```
tag1 + tag2
```
returns a value of 12

```
tag1 * tag2
```
returns a value of 35

```
tag1 - tag2
```
returns a value of –2

```
tag1 / tag2
```
returns a value of 0.71

```
tag1 MOD tag2
```
returns a value of 5

```
tag1 ** tag2
```
returns a value of 78125

Important: Operator precedence is explained later in this chapter.

Relational Operators

Relational operators compare two values to provide a true or false result. If the statement is true, a value of 1 is returned. If false, 0 is returned. There are six relational operators; each has two different symbols.

Table 2.B
The Relational Operators

Symbols	Operator	Example
EQ, =	equal	tag1 = tag2
NE, <>	not equal	tag1 <> tag2
LT, <	less than	tag1 < tag2
GT, >	greater than	tag1 > tag2
LE, <=	less than or equal to	tag1 <= tag2
GE, >=	greater than or equal to	tag1 >= tag2

Example: Relational Operators

For these examples, tag1 = 5 and tag2 = 7.

```
tag1 > tag2
```
is false, so the expression returns a 0

```
tag1 LE tag2
```
is true, so the expression returns a 1

```
tag1 = 5
```
is true, so the expression returns a 1

Logical Operators

Logical operators determine the validity of one or more statements. The operators return a value of 1 if the expression is true, or a value of 0 if false. There are three logical operators: AND, OR, and NOT.

- AND returns a value of 1 if the statement to the right and to the left of the operator are *both* true
- OR returns a value of 1 if *either* the statement to the left or to the right of the operator is true
- NOT reverses the logical value of the statement it operates on

Important: Any statement which evaluates to a non-zero value is regarded as true. For example, the statement "tag1" will be false if the value of tag1 is 0 (zero), and true if tag1 has any other value.

Table 2.C
The Logical Operators

Symbols	Operator	Example
AND, &&	and	(tag1 < tag 2) AND (tag1 = 5)
OR, ||	or	(tag1 = 5) OR (tag1 = 10)
NOT	negation	NOT(tag1 > tag2)

Example: Logical Operators

For these examples, tag1 = 5 and tag2 = 7.

```
(tag1 < tag 2) AND (tag1 = 5)
```
returns a 1 since both statements are true

```
tag1 && tag2
```
returns a 1 since neither tag1 nor tag2 are 0

```
(tag1 > tag2) OR (tag1 = 5)
```
returns a value of 1 since *tag1 = 5* is true

```
NOT(tag1 < tag2)
```
tag1 < tag2 is true and would return a 1 but the NOT operator reverses the logical value, so 0 is returned

Important: The parentheses are essential in the above expressions.

Bitwise Operators

Bitwise operators allow you to examine and manipulate individual bits within a value. These operators can only be applied to integers, not floating point numbers.

Table 2.D
The Six Bitwise Operators:

Symbol	Operator	Example
&	AND	tag1 & 07
|	inclusive OR	tag2 | tag1
^	exclusive OR (XOR)	tag1 ^ 01
>>	right shift	tag1 >> 1
<<	left shift	tag1 << 2
~	complement	~ tag1

The bitwise operators &, |, and ^ compare two integers or tags on a bit by bit basis. The >>, <<, and ~ operators manipulate a single integer or tag.

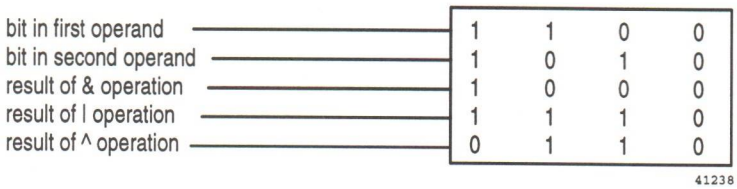
The **bitwise AND (&)** returns an integer with a bit set to 1 if both of the corresponding bits in the original numbers are 1. Otherwise, the resulting bit is 0.

The **bitwise inclusive OR (|)** returns an integer with a bit set to 1 if either or both of the corresponding bits in the original numbers are 1. If both bits are 0, the resulting bit is 0.

The **bitwise exclusive OR (^)** returns an integer with a bit set to 1 if either of the corresponding bits in the original numbers is 1. If both bits are 1 or both are 0, the resulting bit is 0.

The bitwise operators &, |, and ^ are illustrated below:

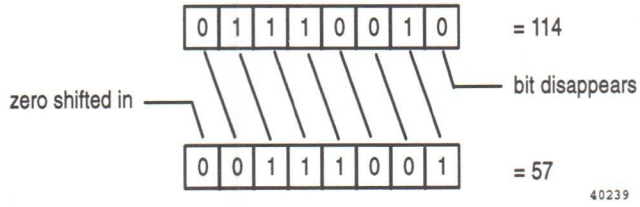
Figure 2.1
The &, |, and ^ Bitwise Operators



The **shift right operator (>>)** shifts the bits within the left operand by the amount specified in the right operand. The bit on the right disappears.

Figure 2.2 shows the result of the expression 114 >> 1 = 57.

Figure 2.2
The Shift Right Operator



Either a 0 or a 1 is shifted in on the left, depending on whether the integer is signed or unsigned. With unsigned integers, 0 is always shifted in on the left; with signed integers, a 0 is shifted in when the number is positive (i.e. the leftmost bit, the sign bit, is 0), and a 1 is shifted in when the number is negative (i.e. the leftmost bit, the sign bit, is one). In other words, with signed integers, the sign of the number is always maintained.

The **shift left operator** (**<<**) shifts the bits within the left operand by the amount specified in the right operand. The bit on the left disappears and a 0 is always shifted in on the right.

The **bitwise complement operator** (**~**) reverses every bit within the number, so that every 1 bit becomes a 0 and vice versa.

Example: Bitwise Operators

For these examples tag1 = 5 (binary 0101), tag2 = 2 (binary 0010)

```
tag1 & tag2
```
returns 0 (binary 0000)

```
tag1 | tag2
```
returns 7 (binary 0111)

```
tag1 ^ tag2
```
returns 7 (binary 0111)

```
tag1 >> 1
```
returns 2 (binary 0010)

```
tag1 << 1
```
returns 10 (binary 1010)

```
~ tag1
```
returns 10 (binary 1010)

Built-in Functions

There are three types of built-in functions:

- functions involving tags
- functions involving the time
- functions involving file operations

These functions return floating point values.

Many of the functions check for specific true/false conditions. Such functions return 1 if the condition is true, and 0 if the condition is false. For instance, the COMM_ERR() function returns 1 if the specified tag has a communication error.

Tag Functions

The following built-in functions examine the status of a tag:

Table 2.E
Tag Functions

This function:	returns this value:
SQRT(*tag*)	the square root of the tag (or of a constant).
COMM_ERR(*tag*)	TRUE if a read or write operation for the specified tag produced a communication failure.
ALM_SUPPRESS(*tag*)	TRUE if the tag's alarms are suppressed.
ALM_FAULT(*tag*)	TRUE if there has been an alarm fault for the specified tag.
ALM_SEVERITY(*tag*)	the severity of the alarm—a value between 1 and 8, or 0 if the tag isn't in alarm.
ALM_LEVEL(*tag*)	the alarm level for the tag: a value between 1 and 8, or 0 if the tag isn't in alarm.
ALM_ACK(*tag*)	TRUE if the tag's alarm has been acknowledged.
ALM_IN_ALARM(*tag*)	TRUE if the tag is in alarm.

Important: The alarm (ALM_) functions will only be relevant if the Alarming option is installed.

Example: Tag Functions

```
ALM_IN_ALARM(vessel3.TIC3.pv)
```
returns 1 if the tag is in alarm or 0 if it isn't.

```
SQRT(vessel3.TIC2.pv)
```
returns the square root of the tag's value.

```
SQRT(25)
```
returns a value of 5.

Time Functions

The following built-in functions examine the system time. These functions use the *time* parameter.

Important: The *time* parameters must be surrounded by quotes.

The *time* parameter can include the following options:

- day of week [Sun, Mon, Tue, Wed, Thu, Fri, or Sat]
- month [Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec]
- date [1 – 31]
- year [1980 – 2100]
- hour of day [00: – 23:]
- minute [:00 – :59]

It doesn't matter what order you list these options in.

Example: Time Parameter

The following all represent the same date and time, and are valid time parameters:

- `"fri aug 18 1989 17:00"`
- `"fri aug 18 1989 17: :00"`
- `":00 aug 18 fri 1989 17:"`

Important: The validity of the date is not checked, so if Aug 18 1989 is *not* a Friday, this error will not be detected.

Example: Incomplete Time Parameter

The following are valid examples of incomplete time parameters:

- `"17:00"`
  means once a day at 5:00 pm
- `":30 "`
  means once an hour, on the half hour
- `"fri 17:"`
  means 5:00 pm each Friday

Table 2.F
Time Functions

This function:	returns this value:
TIME("*time*")	TRUE if the time specified is the current time.
BEFORE_TIME("*time*")	TRUE if the expression is evaluated before the specified time.
AFTER_TIME("*time*")	TRUE if the expression is evaluated after the specified time.

Example: Incomplete Time Parameter

```
TIME("sun  feb  18  1990  14:30")
```

returns 1 if it is exactly 2:30 pm on Sunday Feb 18, 1990; otherwise 0 is returned.

Important: If the Evaluation Period is greater than 60 seconds, this function may never be evaluated as true.

```
AFTER_TIME("sun feb 18 1990  14: :30")
```

returns 1 the first time the expression is evaluated after 2:30 pm on Sunday Feb 18, 1990 and will continue returning 1 from this point on.

```
BEFORE_TIME("Feb 18 1990")
```

returns 1 if it is not yet Feb 18, 1990; returns 0 if the date is on or after Feb 18, 1990.

File Functions

The following built-in functions determine whether a specific file exists, and how much free space there is on a disk.

The *file* parameter is the DOS pathname, surrounded by quotes.

The *drive* parameter is the letter of the drive.

Table 2.G
File Functions

This function:	returns this value:
FILE_EXISTS("*file*")	TRUE if the specified file exists.
FREE_BYTES(*drive*)	the number of bytes free on the specified drive.

Examples: File Functions

```
FILE_EXISTS("e:\access\dat\activity.000")
```
returns 1 if the file exists or 0 if the file doesn't exist. You may want to use this for setting tags when a specified file has been created or deleted.

```
FREE_BYTES(c)
```
returns the number of bytes available on drive C. You may want to display it on a system graphic or to create an alarm when disk space is getting low.

Operator Precedence

There are three rules used to determine the order for evaluating an expression that has more than one operator.

1. The operator with the highest precedence is evaluated first.
2. When two operators have equal precedence, they are evaluated from left to right.
3. To change the order of precedence, use parentheses.

Here are the operators from highest to lowest precedence:

Table 2.H
Operator Precedence

Precedence Level	Name	Symbols
1 (Highest)	parentheses	( )
2	Logical negation Bitwise complement	NOT ~
3	multiplication division modulus (remainder) exponent logical and bitwise and bitwise shift right bitwise shift left	* / MOD, % ** AND, && & >> <<
4	addition subtraction logical or bitwise inclusive or bitwise exclusive or the built-in functions	+ – OR, || | ^
5 (lowest)	equal not equal less than greater than less than or equal to greater than or equal to	EQ, = NE, <> LT, < GT, > LE, <= GE, >=

Example: Operator Precedence

For these examples, tag1 = 5, tag2 = 7 and tag3 =10.

```
(tag1 > tag2) AND (tag1 < tag 3)
```

is evaluated in this sequence:

1. tag1 > tag2 = 0
2. tag 1 < tag3 = 1
3. 0 AND 1 = 0

The expression evaluates to 0.

```
tag1>tag2 AND tag3
```

is evaluated in this sequence:

1. tag2 AND tag3 = 1
2. tag1 > 1 = 1

The expression evaluates to 1.

```
NOT tag1 AND tag2 > tag3 ** 2
```

is evaluated in this sequence:

1. NOT tag1 = 0
2. 0 AND tag2 = 0
3. tag3 ** 2 = 100
4. 0 > 100 = 0

The expression evaluates to 0.

IF-THEN-ELSE

The IF-THEN-ELSE structure allows an expression to make a decision.

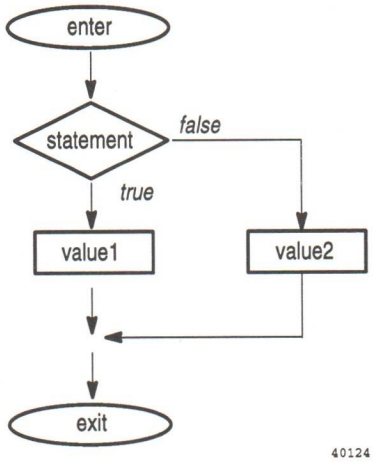
The format of the IF-THEN-ELSE structure is:

IF *statement* THEN *value1* ELSE *value2*

If the *statement* is true then the expression will return *value1*; if the *statement* is false then the expression returns *value2*. Keep in mind that the *statement* is a mathematical equation and that true means a non-zero value, and false means zero.

The IF-THEN-ELSE structure is illustrated below.

Figure 2.3
The IF-THEN-ELSE Structure



Nested IF-THEN-ELSE structure

It is common to nest an entire IF-THEN-ELSE structure inside another IF-THEN-ELSE structure.

When nesting IF-THEN-ELSE structures, it is important to remember that an ELSE is associated with the last IF that doesn't have its own ELSE.

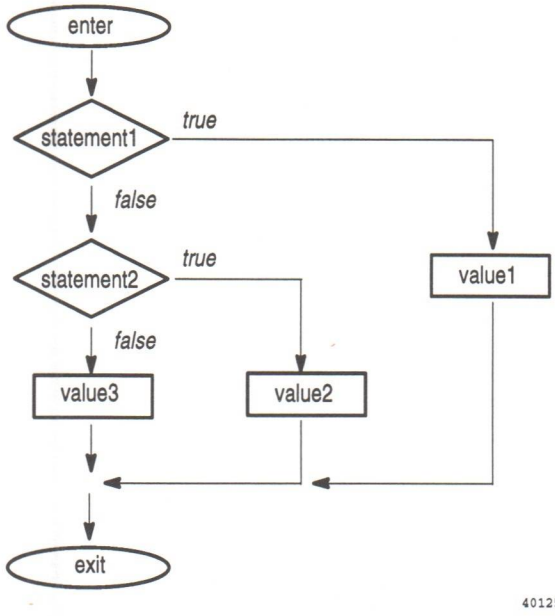
Example 1: Nested IF-THEN-ELSE

This expression:

```
IF (statement1) THEN (value1)
ELSE IF (statement2) THEN (value2)
           ELSE (value3)
```

has this interpretation:

Figure 2.4
Nested IF-THEN-ELSE



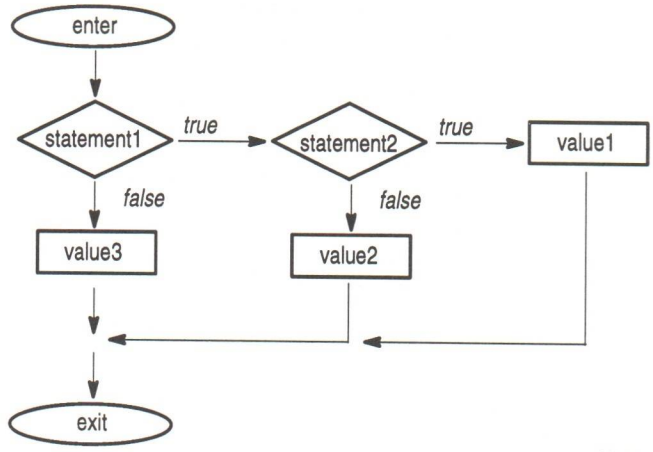
Example 2: Nested IF-THEN-ELSE

This expression:

```
IF (statement1) THEN
          IF (statement2) THEN (value1)
          ELSE (value2)
ELSE (value3)
```

has this interpretation:

Figure 2.5
Nested IF-THEN-ELSE



Comments

Comments begin with an exclamation point. All text following the exclamation point, to the end of the line, is not evaluated.

Example: Comments in a ControlView Expression

In a single line such as:

```
IF a>b THEN c ELSE d ! if/then expression
```

only that part preceding the exclamation point is evaluated.

In several lines such as:

```
IF a>b ! temp a greater than temp b
    THEN c ! shut valve
ELSE d ! do nothing
```

only those parts preceding the exclamation points in each line are evaluated.

Expression Formatting

The derived tag editor ignores tabs, new line characters, and multiple spaces, so a large expression can be condensed to make maximum use of the editor space.

Example: Expression Formatting

The following expression:

```
If (tag1 > tag2) then 0
else if (tag1 > tag 3) then 2
          else 4
```

Could be condensed to the following:

```
If (tag1 > tag2) then 0 else if (tag1 > tag3) then
2 else 4
```

Important: Do not allow tag names, keywords, function names or function arguments to span more than one line.

Appendix A

Derived Tags Commands

DERIVED

DERIVED [*file*] [*/n*]

Calls up the Derived Tag Editor, which is used to create or modify a file of derived tags. The parameters are:

[*file*]	Is the name of the derived tag file (without a file extension). If you omit [*file*], the Editor loads the default file DTAGS.
[*/n*]	Turns autoverification off while the file is loading. When autoverification is on, the editor checks that the tag names used in the derived tag file, actually exists in the CVD.

Examples: DERIVED Command

```
DERIVED dtg2
```

Opens the Derived Tags Editor containing the file dtg2, and verifies that the tags mentioned in the file exist in the loaded database.

```
DERIVED dtg3 /n
```

Opens the Derived Tags Editor containing the file dtg3, without verifying that the tags mentioned in the file exist in the loaded database.

```
DERIVED
```

Starts up the Derived Tags Editor with the default file, DTAGS.

DERIVEDOFF

DERIVEDOFF

Stops the derived tag processor and unloads the derived tag file.

DERIVEDON

DERIVEDON [*file*]

Loads a derived tag file and starts the derived tag processor. The parameter is:

[*file*]	Is the name of the derived tag file (without a file extension). If you omit [*file*], the default file, DTAGS, is loaded.

Example: DERIVEDON Command

```
DERIVEDON DTG2
```

Loads the derived tag file DTG2, and begins processing that file.

Important: A database must be loaded before the processor is started and only one derived tag file may be processed at one time.

ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

As a subsidiary of Rockwell International, one of the world's largest technology companies — Allen-Bradley meets today's challenges of industrial automation with over 85 years of practical plant-floor experience. More than 11,000 employees throughout the world design, manufacture and apply a wide range of control and automation products and supporting services to help our customers continuously improve quality, productivity and time to market. These products and services not only control individual machines but integrate the manufacturing process, while providing access to vital plant floor data that can be used to support decision-making throughout the enterprise.

With offices in major cities worldwide

WORLD HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (1) 414 382-2000
Telex: 43 11 016
FAX: (1) 414 382-4444

EUROPE/MIDDLE EAST/AFRICA HEADQUARTERS
Allen-Bradley Europe B.V.
Amsterdamseweg 15
1422 AC Uithoorn
The Netherlands
Tel: (31) 2975/43500
Telex: (844) 18042
FAX: (31) 2975/60222

ASIA/PACIFIC HEADQUARTERS
Allen-Bradley (Hong Kong) Limited
Room 1006, Block B, Sea View Estate
28 Watson Road
Hong Kong
Tel: (852) 887-4788
Telex: (780) 64347
FAX: (852) 510-9436

CANADA HEADQUARTERS
Allen-Bradley Canada Limited
135 Dundas Street
Cambridge, Ontario N1R 5X1
Canada
Tel: (1) 519 623-1810
FAX: (1) 519 623-8930

LATIN AMERICA HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (1) 414 382-2000
Telex: 43 11 016
FAX: (1) 414 382-2400

Publication 6190-6.5.11 November 1992
Supersedes Publication 6190-6.5.11,—August 1991

PN 504629904